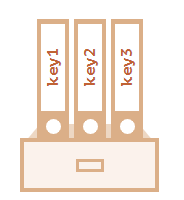
**Объекты**

Объект может быть создан с помощью фигурных скобок {…} с необязательным списком свойств. Свойство – это пара «ключ: значение», где ключ – это строка или символ (также называемая «именем свойства»), а значение может быть чем угодно.

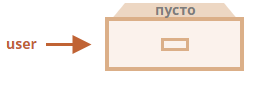
можем представить объект в виде ящика с подписанными папками. Каждый элемент данных хранится в своей папке, на которой написан ключ. По ключу папку легко найти, удалить или добавить в неё что-либо.



Пустой объект («пустой ящик») можно создать, используя один из двух вариантов синтаксиса:

let user = new Object(); // синтаксис "конструктор объекта"

let user = {}; // синтаксис "литерал объекта" или литеральная нотация.



## [Литералы и свойства](https://learn.javascript.ru/object" \l "literaly-i-svoystva)

При использовании литерального синтаксиса {...} можно сразу поместить в объект несколько свойств в виде пар «ключ: значение»:

let user = { // объект

name: "John", // под ключом "name" хранится значение "John"

age: 30 // под ключом "age" хранится значение 30

};

У каждого свойства есть ключ (также называемый «имя» или «идентификатор»)

обращение к свойствам

alert( user.name ); // John

Значение может быть любого типа.

добавить свойство

user.isAdmin = true;

удаления свойства оператором delete

delete user.age;

Если имя свойства состоит из нескольких слов, но тогда оно должно быть заключено в кавычки:

let user = {

name: "John",

age: 30,

"likes birds": true // имя свойства из нескольких слов должно быть в кавычках

};

Когда последнее свойство заканчивается запятой, это называется «висячая запятая». Такой подход упрощает добавление, удаление и перемещение свойств, так как все строки объекта становятся одинаковыми.

**Объект, объявленный как константа, может быть изменён**

const user = {

name: "John"

};

user.name = "Pete"; // (\*)

alert(user.name); // Pete

объявление const защищает от изменений только саму переменную user, а не её содержимое.

Определение const выдаст ошибку только если мы присвоим переменной другое значение: user=....

Чтобы получить доступ к свойству, мы можем использовать:

* Запись через точку: obj.property.
* Квадратные скобки obj["property"]. Квадратные скобки позволяют взять ключ из переменной, например, obj[varWithKey].

## [Квадратные скобки](https://learn.javascript.ru/object" \l "kvadratnye-skobki)

Для свойств, имена которых состоят из нескольких слов, доступ к значению «через точку» не работает:

// это вызовет синтаксическую ошибку

user.likes birds = true

JavaScript видит, что мы обращаемся к свойству user.likes, а затем идёт непонятное слово birds. В итоге синтаксическая ошибка.

Точка требует, чтобы ключ был именован по правилам именования переменных. То есть не имел пробелов, не начинался с цифры и не содержал специальные символы, кроме $ и \_.

Для таких случаев существует альтернативный способ доступа к свойствам через квадратные скобки. Такой способ сработает с любым именем свойства:

let user = {};

// присваивание значения свойству

user["likes birds"] = true;

// получение значения свойства

alert(user["likes birds"]); // true

// удаление свойства

delete user["likes birds"];

Квадратные скобки также позволяют обратиться к свойству, имя которого может быть результатом выражения. Например, имя свойства может храниться в переменной:

let key = "likes birds";

// то же самое, что и user["likes birds"] = true;

user[key] = true;

Здесь переменная key может быть вычислена во время выполнения кода или зависеть от пользовательского ввода. После этого мы используем её для доступа к свойству. Это даёт нам большую гибкость.

Пример:

let user = {

name: "John",

age: 30

};

let key = prompt("Что вы хотите узнать о пользователе?", "name");

// доступ к свойству через переменную

alert( user[key] ); // John (если ввели "name")

Запись «через точку» такого не позволяет:

let user = {

name: "John",

age: 30

};

let key = "name";

alert( user.key ); // undefined

### [Вычисляемые свойства](https://learn.javascript.ru/object" \l "vychislyaemye-svoystva)

Квадратные скобки используются в литеральной нотации для создания вычисляемого свойства.

Пример:

let fruit = prompt("Какой фрукт купить?", "apple");

let bag = {

[fruit]: 5, // имя свойства будет взято из переменной fruit

};

alert( bag.apple ); // 5, если fruit="apple"

Запись [fruit] означает, что имя свойства необходимо взять из переменной fruit.

И если посетитель введёт слово "apple", то в объекте bag теперь будет лежать свойство {apple: 5}.

Этот код аналогичен коду выше

let fruit = prompt("Какой фрукт купить?", "apple");

let bag = {};

// имя свойства будет взято из переменной fruit

bag[fruit] = 5;

В квадратных скобках можно использовать более сложные выражения:

let fruit = 'apple';

let bag = {

[fruit + 'Computers']: 5 // bag.appleComputers = 5

};

Квадратные скобки дают намного больше возможностей, чем запись через точку. Они позволяют использовать любые имена свойств и переменные, хотя и требуют более громоздких конструкций кода.

Подведём итог: в большинстве случаев, когда имена свойств известны и просты, используется запись через точку. Если же нам нужно что-то более сложное, то мы используем квадратные скобки.

## [Свойство из переменной](https://learn.javascript.ru/object" \l "svoystvo-iz-peremennoy)

Часто необходимо использовать существующие переменные как значения для свойств с тем же именем.

Например:

function makeUser(name, age) {

return {

name: name,

age: age

// ...другие свойства

};

}

let user = makeUser("John", 30);

alert(user.name); // John

В примере выше название свойств name и age совпадают с названиями переменных, которые подставляются в качестве значений этих свойств. Такой подход настолько распространён, что существуют специальные короткие свойства для упрощения этой записи.

function makeUser(name, age) {

return {

name, // то же самое, что и name: name

age // то же самое, что и age: age

// ...

};

Обычные и короткие свойства, можно комбинировать в одном объекте:

let user = {

name, // тоже самое, что и name:name

age: 30

};

## [Ограничения на имена свойств](https://learn.javascript.ru/object" \l "ogranicheniya-na-imena-svoystv)

нет никаких ограничений к именам свойств. Они могут быть в виде строк или символов (специальный тип для идентификаторов).

Все другие типы данных будут автоматически преобразованы к строке. Например, если использовать число 0 в качестве ключа, то оно превратится в строку "0":

let obj = {

0: "Тест" // то же самое что и "0": "Тест"

};

// обе функции alert выведут одно и то же свойство (число 0 преобразуется в строку "0")

alert( obj["0"] ); // Тест

alert( obj[0] ); // Тест (то же свойство)

## [Проверка существования свойства, оператор «in»](https://learn.javascript.ru/object" \l "proverka-suschestvovaniya-svoystva-operator-in)

можно получить доступ к любому свойству. Даже если свойства не существует – ошибки не будет!

При обращении к свойству, которого нет, возвращается undefined. Это позволяет просто проверить существование свойства:

let user = {};

alert( user.noSuchProperty === undefined ); // true означает "свойства нет"

Также существует специальный оператор "in" для проверки существования свойства в объекте.

Синтаксис оператора:

"key" in object

Пример:

let user = { name: "John", age: 30 };

alert( "age" in user ); // true, user.age существует

alert( "blabla" in user ); // false, user.blabla не существует

Обратите внимание, что слева от оператора in должно быть *имя свойства*. Обычно это строка в кавычках.

Если мы опускаем кавычки, это значит, что мы указываем переменную, в которой находится имя свойства. Например:

let user = { age: 30 };

let key = "age";

alert( key in user ); // true, имя свойства было взято из переменной key

Для чего вообще нужен оператор in? Разве недостаточно сравнения с undefined?

В большинстве случаев прекрасно сработает сравнение с undefined. Но не когда свойство существует, но содержит значение undefined:

let obj = {

test: undefined

};

alert( obj.test ); // выведет undefined, значит свойство не существует?

alert( "test" in obj ); // true, свойство существует!

В примере выше свойство obj.test технически существует в объекте. Оператор in сработал правильно.

Подобные ситуации случаются очень редко, так как undefined обычно явно не присваивается. Для «неизвестных» или «пустых» свойств мы используем значение null.

## [Цикл "for..in"](https://learn.javascript.ru/object" \l "forin)

Для перебора всех свойств объекта используется цикл for..in.

Синтаксис:

for (key in object) {

// тело цикла выполняется для каждого свойства объекта

}

### [Упорядочение свойств объекта](https://learn.javascript.ru/object" \l "uporyadochenie-svoystv-obekta)

Свойства упорядочены особым образом: свойства с целочисленными ключами сортируются по возрастанию, остальные располагаются в порядке создания.

let codes = {

"49": "Германия",

"41": "Швейцария",

"44": "Великобритания",

// ..,

"1": "США"

};

for (let code in codes) {

alert(code); // 1, 41, 44, 49

}

…С другой стороны, если ключи не целочисленные, то они перебираются в порядке создания

**Целочисленные свойства? Это что?**

Термин «целочисленное свойство» означает строку, которая может быть преобразована в целое число и обратно без изменений.

То есть, "49" – это целочисленное имя свойства, потому что если его преобразовать в целое число, а затем обратно в строку, то оно не изменится. А вот свойства "+49" или "1.2" таковыми не являются: